Synthesizing UML Statecharts From Requirements Scenarios + Propositional Constraints

— Position Statement —

Jon Whittle
QSS Inc.
NASA Ames Research Center
Moffett Field, CA, 94035

The Unified Modeling Language (UML) (OMG 2001) provides a standardized collection of notations for describing artifacts in a software-intensive system. Each UML notation represents a particular viewpoint of a software (sub-)system. For example, UML sequence diagrams describe system behavior in terms of the interaction scenarios between multiple objects. UML statecharts, on the other hand, describe the behavior of a single object. The UML constraint language, OCL, (Warmer & Kleppe 1999), describes model entities in a declarative fashion. Software developers typically use a variety of notations for behavioral modeling but currently have no way to maintain the consistency between viewpoints. Ultimately, if behavioral models are going to be used in simulation or implementation, the various notations must be merged together. (Whittle & Schumann 2000) presents an algorithm for synthesizing UML statecharts from a (possibly conflicting) set of scenarios (sequence diagrams) and propositional OCL constraints. This algorithm is a step towards the goal of semi-automated merging of model viewpoints.

The importance of generating behavioral descriptions directly from expected scenarios of system behavior was noted as early as 1987 in Harel's original paper on statecharts (Harel 1987):

> *many of the people that were involved in the avionics project* [. . .] *were able to state many desirable scenarios, such as firing a missile or updating the aircraft's location, in precise detail* [. . .] *it would seem beneficial to be able to derive a* **reasonable** *statechart description from a large set of scenarios* [. . .].

Despite recent work on this synthesis problem (see e.g., (Systä 2000; Harel & Kugler 2000; Uchitel, Kramer, & Magee 2001)), only limited consideration has been given to what kind of synthesis machinery can produce a *reasonable* statechart. Since scenarios are in general partial, the generated statechart will most likely be modified by the system designer. Hence, it must be concise and readable. (Whittle & Schumann 2000) addresses this question of what makes a "good" synthesis engine by incorporating the following points:

- scenarios may be inconsistent with each other — any synthesis engine should detect (at least partially) these inconsistencies;
- scenarios will likely duplicate behaviors — these should be merged during synthesis, thus leading to an optimized, concise statechart representation;
- generated statecharts should be readable — in terms of statecharts, this implies the introduction of hierarchy;
- generated statecharts will be hand modified — the synthesis machine should support iterative refinements by checking the original scenarios against the modified statecharts.

The algorithm in (Whittle & Schumann 2000) first compares each scenario (i.e., sequence diagram) to the propositional constraints. Pre/post conditions on the events in the scenario can be used to check that event sequences are valid. Each event is annotated with partial information from these pre/post conditions which is then propagated throughout the diagram using the frame axiom. This results in an annotated sequence diagram which is then translated into a set of finite state machines, one for each scenario object. Once all sequence diagrams have been considered, the state machines for each object are merged using the annotations as a guide to merge states where possible. Hierarchy can be introduced automatically into the resulting state machines by partitioning the states according to instantiations of significant mode variables.

A prototype for this algorithm has been implemented in Java and has been linked to commercial UML tools using XML as an interchange format. Case studies are being undertaken on examples from air traffic control software currently under development at NASA Ames.

Acknowledgements

Johann Schumann was involved in the development and implementation of the algorithm. Jyoti Saboo wrote the XML interface.

References

Harel, D., and Kugler, H. 2000. Synthesizing state-based object systems from LSC specifications. In *Fifth International Conference on Implementation and Application of*

Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Automata (CIAA2000), Lecture Notes in Computer Science. Springer-Verlag.

Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Science of Computer Programming* 8:231–274.

2001. Unified Modeling Language specification version 1.4. Available from The Object Management Group (http://www.omg.org).

Systä, T. 2000. Incremental construction of dynamic models for object oriented software systems. *Journal of Object Oriented Programming* 13(5):18–27.

Uchitel, S.; Kramer, J.; and Magee, J. 2001. Detecting implied scenarios in message sequence chart specifications. In *Proceedings of the 9th European Software Engineering Conference (ESEC01)*.

Warmer, J., and Kleppe, A. 1999. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley Object Technology Series. Addison-Wesley.

Whittle, J., and Schumann, J. 2000. Generating Statechart Designs From Scenarios. In *Proceedings of International Conference on Software Engineeering (ICSE 2000)*, 314–323.